Final Report of Senior Design Project
Winter/Spring 2002

The University of California at Riverside
Department of Electrical Engineering

Machine Vision Interface With Seiko Robot Arm II

Prepared by: Zhi Xu and Ghazi Ting

Technical Faculty Advisors: Mr. Dan Giles, Principal Electronics Technician
Dr. Gerardo Beni, Electrical Engineering Faculty of the University of California at Riverside

This project is support by the Electrical Engineering Department of the University of California at Riverside

Submitted on June 7, 2002 at 5:00 PM

Executive Summary – The objective of this project is to create a third party vision system to be utilized by the Seiko D-TRAN RT3200 robot arm that belongs to the Electrical Engineering Department of UCR. Working with the Visual Basic 6.0 platform started by last year's senior design group, the industrial image processing software VisionBlox is used to design a vision system capable of live image. A user-friendly interface has been developed to integrate the Seiko robot with the vision system so the two components can operate together. In addition, with the interface users can control the robot and use the vision system without any prior knowledge of DARL commands or VisionBlox. With the aid of the vision system, the arm is able to perform regular industrial type task such as performance check, object detection, and objection recognition automatically. The developed vision system has made the robot much more user friendly and its capabilities have increased tremendously. An updated robot was developed by Seiko model D-TRAN RT330 that has the vision feature, which shows the importance of our project and the real life applicability of a vision system for the RT3200 robot arm.

Keywords – robotics, vision system, serial port communications, automation, object detection

Table of Contents

Chapter 1 ~ Introduction

1.1 Introduction

The driving force for this design is to fulfill the senior design course requirements for the Electrical Engineering Department at the University of California, Riverside. For two quarters, every undergraduate student in the Electrical Engineering major at the University of California, Riverside must present a design project to the Electrical Engineering Department prior to graduation. The undergraduate student has the freedom to select any design project as he or she pleases, but with the approval of the advising professors. For our senior design project, we decided to continue the Machine Vision Interface with Seiko Robot Arm project started by prior undergraduate students at University of California, Riverside. The main objective is to develop a machine vision interface for the Seiko RT3200 Robot Arm in one of the UCR's Electrical Engineering Department Laboratory. The Seiko RT3200 Robot Arm is an industrial robot that can be used for many applications, and by adding a vision system to the robot will greatly improve its capabilities and value. Also, once the vision system is complete, it can be marketed and implemented for other Seiko RT3200 Robot Arm that needs a vision system. Section 1.2 talks about the Historic Background of the robotic industry and section 1.3 contains a Glossary of Acronyms and Abbreviations important to this report.

1.2 Historic Background

The Seiko RT3200 Robot arm is an industrial robot, and it is of the cylindrical type. It is a hybrid mix that incorporates the rotary motion of the SCARA and the linear motion of a CARTESIAN. There are four axis of movement for the robot. (x, y, z, a). The x-axis and the y-axis take care of the flat surface of the robot's movement while z-axis maps the height of the robot. The a-axis is rotational movement of the end-effector on the robot. The RT3200 has a standard cycle time of 0.8 seconds that the robot can work in, a very high speed compare to any other kind of robot. DARL is the original image developed by Seiko for the RT3200 model robot, which is very similar to the programming language BASIC.

1.3 A Glossary of Acronyms and Abbreviations

UCR – University of California, Riverside
DARL – D-Tran Assembly Robot Language
Visual Basic – Computer programming language that runs in Microsoft Window
End-effectors – The tool at the end of the Seiko RT3200 robot used to perform specific tasks.
Seiko RT3200 – Industrial robot used in the production industry to perform assembly type jobs.
Editable Image – This is the display of image entire image on the user interface
Train Image – The small region within the editable image use to select a specific object out of the whole image

Chapter 2 ~ Design and Technical Formulation

2.1 Introduction

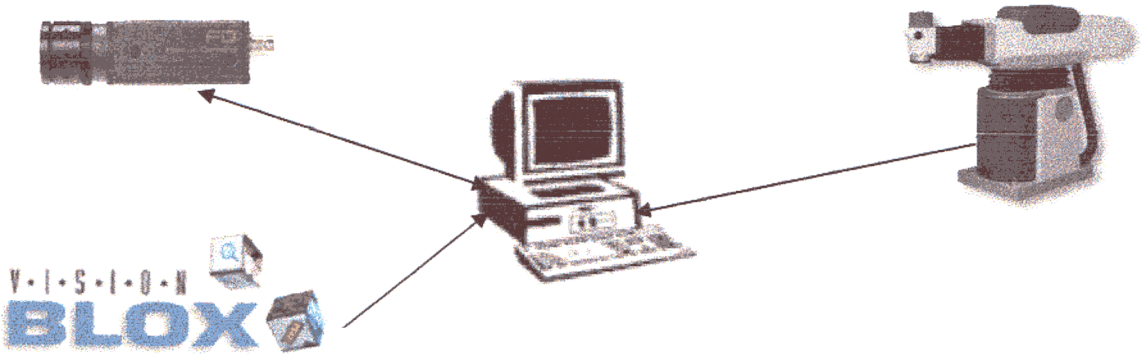
The design approaches selected for this project make use of the work completed from last year's senior design group and improve on their work. The main part of the design deals with controlling the Seiko D-TRAN robot in Visual Basic and incorporating a third party vision system to be used by the Seiko robot. Section 2.2 will contains the Problem Statement and section 2.3 is the Design Specifications. In sections 2.4 and 2.5, the focus is on the Design Approach and the Alternative Approaches to the project.

2.2 Problem Statement

The goal of this project is to upgrade the Seiko RT3200 so that it utilizes a third party vision system. The problem includes creating vision software that can incorporate the already existing VisionBlox software with the Seiko robot arm. In addition, the vision system needs to be able to recognize some trained objects and perform particular task with the object. Moreover, the vision system needs to be able demonstrate some automation procedure with the robot arm.

2.3 Design Specifications

The specification for our design requires the vision system to be able to capture and analyze an image in real time. The robot also needs to be able to be controlled through a personal computer. Using the PC, a vision system must be incorporated into the Seiko RT3200 robot. The vision system that needs to be created have to be able to recognize an object, get the coordinates of the found object, and then send the coordinates of the object to the robot so it can interact with it. Automation between the vision system and the Seiko robot arm is desirable feature of the design project.



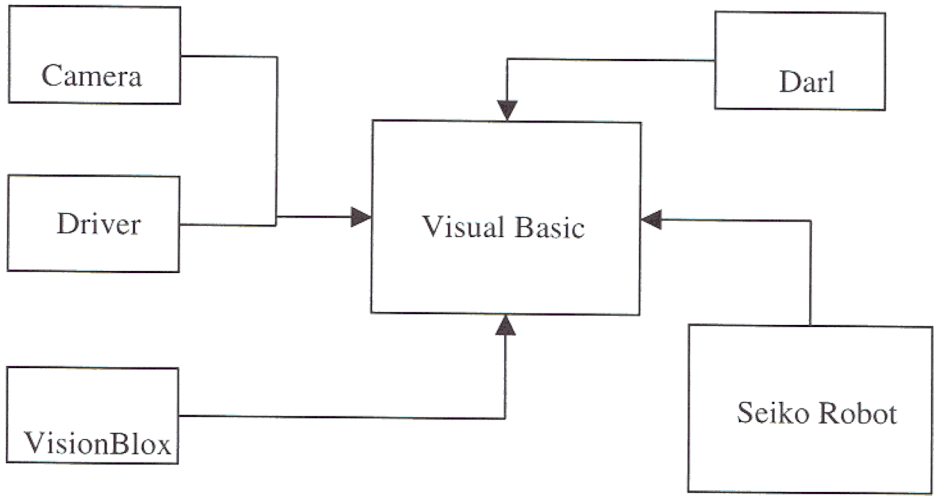


Fig. 2.2 Technical Design

2.4 Design Approach

The approach taken to fulfill the specifications of our senior design project is to continue with the development of last year's senior design group. In the preceding year, a group of seniors in the University of California, Riverside Electrical Program started the implementation of vision for the Seiko robot arm. They were able to utilize Visual Basic 6.0 to incorporate the vision software VisionBlox and the Seiko robot language Darl to work together as one entity. With parts of the machine vision interface already developed, we needed only to continue the development by upgrading the capabilities of the vision system, creating a more user friendly interface, and making the Seiko D-TRAN RT3200 Robot Arm have a totally automated vision system.

2.5 Alternative Approaches

Another possible approach could have been to start the project fresh and not use the codes provided by last year's group. The advantage would be that we will have total control of the decisions of the project and any errors or issues that arises we would be able to answer. The drawback to this approach is that we would be reproducing many of the same information that is already available and the difference from continuing from last year's end point versus starting anew might be minimal. Other possible alternative approaches can be found in section 3.3 of the report.

Chapter 3 ~ Design Evaluation

3.1 Introduction

The most important measure of the effectiveness or how good the design of this project is deals with the compatibility and accuracy between the robot and the vision system. Therefore, most of the test were focused on the vision system being able to identify and differentiate objects in an image and determine whether or not the trained object is in the image. Next, how accurate is the robot in going to location of the desired object specified by the vision system. From the results of our tests, we conclude that are design has achieved the satisfactory level that we desire. Section 3.2 is the Test Plan and Result, and section 3.3 is the Design Comparison / Design Trade Off.

3.2 Test Plan and Result

- Delay needed when moving robot with VB
- Sending commands using VB through serial port
- Capturing and storing of live image
- Parameters for comparing trained with live image
- Pixel to world coordinate system calculation
- Accuracy of the vision system in real time
- The repeatability of the robot
- The capability of the vision system
- A compatible speed between robot and vision
- Ability to perform multiple task simultaneously

(Vision)

Using the FeatureFindTool in VisionBlox to compare an object.

Utilized the save command from the Controller form to store an image and then wrote a load function in the Commands form to load the saved images saved. Next, the FeatureFindTool.Found function was used to return the comparison result and displayed in a text box. The FeatureFindTool compares everything within the trained region, so if the same object is placed on a different background from the trained image, FeatureFindTool.Found will likely to return a False. Setting the minimum % on the form will change the FeatureFindTool.MinScore and this can make some difference. Test shows that 75% works well for objects with the same object type. The test we used is two washers.

Testing for the MinScore Value:

Comparing two washers . (Washer A is the washer in the trained image).

Original image:	Washer A	**True** (0 - 100)	**False** none
	Washer B -	**True** (0 – 89)	**False** (90 – 100)
Image 2	Washer A	**True** (0 – 87)	**False** (88 – 100)
	Washer B -	**True** (0 – 88)	**False** (89 – 100)
Image 3	Washer A -	**True** (0 – 84)	**False** (85 – 100)
	Washer C - (smaller washer)	**True** (0 – 34)	**False** (35 – 100)

Training an image

The edshpTrain.Load and edshpTrain.Save is used to load and save an image in the editable shape region of the editable image. To use the FeatureFindTool.Found function, the image must be first trained. The edshpInspection.save and FeatureFindTool.save should always be used whenever the edshpTrain.Save command is used. The same for the edshpTrain.Load command. It appears that the CaliperTool commands works the same way as the FeatureFindTool, but there were errors when we try to incorporate them into our program, mainly when training the CaliperTool.

Saving and Loading an image.

Mainly to store and load the image in the editable image and the editable train shape region. It is possible to load an editable image and a trained image not in the editable image.

Getting coordinates when a trained image is found in the editable image.

The commands FeatureFindTool.ResultX and FeatureFindTool.ResultY is used to get the center position of the trained image with respect to the editable image when FeatureFindTool.Found returns a True. The top left corner of the editable image is (0,0) and the bottom right corner is (640,480). These are pixel coordinates and a calculation will be needed to convert the coordinates into spatial coordinates so the Seiko Robot can utilize it. More details on the conversion from pixel coordinates to world

coordinate in the Robotic test section.

Displaying live images on the editable image

The image captured by the frame grabber must be stored onto a temporary variable and then using the temporary variable to have it display on the editable image. Look at the Private Sub RunProg( ) function to see how this is done. I tried utilize the OCX file for the frame grabber to set as the video card and reference the image from the frame grabber to the editable image directly without first storing it in a temporary image. However, I was unable to reference frame grabber card in my code and had use the alternate option.

Automating the process of displaying the live images

To automate the process, I created a Do-While a loop to continue reference the frame grabber image to a temporary image and compare it with the stored trained image. If a trained image is found, then the process stops and the Pick( ) function is called. At first, I thought the loop I created was causing the computer to freeze, but after placing a variable to count the loops and setting a limit on how much the Private Sub RunProg( ) loops, the freeze seems to come from the infinite loop of the image always returning False.

(Robotic)

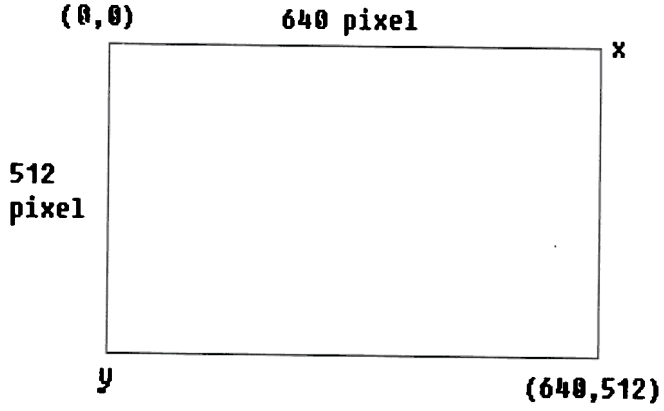


Fig. 3.1 Pixel Coordinate

Testing movement limitation of the robot arm

Home position is 300. 0. 0. 0.
X is from 300 to 600
Y is from –277.615 to +277.615
Z is from –120 to 0
A is from –100degree to 100 (for the end-effector)

For all movement of the robot, we first programmed and tested it in DARL first and then after it works, the code is incorporated into the Visual Basic code.

Testing for delay on the robot

First we set the robot at speed 50, while teaching the robot to learn a new position a small delay is needed, and we set it to 5. When I try to move the robot arm from one position to another position, another delay is needed to allow the robot time to get to the new position. The delay was first set to 50, which is too short and the robot sometimes will skip commands. So the delay has been set to 100 now it works fine.

Then we try to set the robot at full speed 300, between teaching the robot to learn a new position the delay was set to 5. When we try to move the robot arm from one position to another, the delay is set to 40 and it works fine. Shorter delay will work for shorter distance traveled by the robot, but delay 40 is a safe and reasonable delay time.

Delay function used is actually a for-loop that just loops multiple times. Delay of 1 is equivalent to 1,000,000 loops.

Converting pixel coordinate system to world coordinate system

When camera to ground height is 450mm, the screen is 110mm X 90m and pixel to mm conversion is 1:5.8mm. When we try to use the conversion to get the robot to the converted location at this height, it is always off. We found out that the camera is not mounted at the center of the robot and is the cause of our conversion to be off.

Later, we found at the 450mm height, the image is out of focus. So the new height that the image is in focus and the end-effector can still pick up object is at the height of 340mm. At this height, the screen is 95mm X 67mm and the pixel to mm conversion is 1:6.737. At each height, to test for screen size and unit conversion, four corners are drawn on a paper at the size of the screen. Displayed on the screen, and using image size on screen and the size of the four corners, the new unit conversion is determined. To test, object is set at different location and the robot will try going to the object. Many trials are taken to test for accuracy.

3.3 Design Comparison / Design Trade Off

There are two major changes that we made to our original design plan, use of the CaliperTool feature in VisionBlox and the workspace used for our automation system. Originally, before fully understanding the capabilities and functionality of the FeatureFindTool of VisionBlox, we were considering adding the CaliperTool component to our vision system. We assumed that without some sort of way to separate the foreground from the background, there will be no way to detect an object. However, since we can control the background, we forced the background of all images we work with to be always white or black. With the ability to control the background, then just by using the FeatureFindTool we are able to utilize the features of an object alone to distinguish one object from another.

Another major change we made to our original design was the workspace for the robot when we automate our vision system. Originally, we were planning to have a box shaped workspace for our vision system and robot would scan for objects within that workspace. To do this would require some extremely difficult algorithms to compute the translation from the pixel coordinate system to the world coordinate system, however, so we simplified the process by having the robot just scan forward. To have the workspace of a box would require the robot and camera to move sideways, which can throw off the camera orientation with respect to the robot since the camera is mounted on the robot. For when the robot moves in the y direction, the whole camera is rotated by some inconsistent degree, which would be difficult to calculate the relationship between the robot movements with respect to the angle change of the camera. By not having the robot being able to move sideways limits the workspace in which our robot can work with in our automation mode, it simplified our calculation of translating pixel locations into world locations that the Seiko robot can use.

Chapter 4 ~ Administration

4.1 Introduction

Our project is a continuation of an existing project; therefore we did not spend any money for this project. The Budget/Cost Analysis of section 4.2 contains prices of what it would have cost had we had to buy all the parts ourselves. Section 4.3 talks about the Marketability of the design we help develop.

4.2 Budget/Cost Analysis

Seiko D-TRAN RT3200		$ 10,000
Vision1 Mono-kit:		
• Hitachi KP-M2 Camera		
• Imagenation PX610 PCI Frame-grabber Card		
• VisionBlox software		
• 12 volt DC Power Supply & Cables		$1,975
Computar C-Mount, 25mm, f1.3 lens		$128
Visual Basic 6.0		$99
Development cost:		
• 200 hours x $100 per hour		$20,000
	Total:	$32,202
Purchasing newer model:		
Seiko RT 3300		$60,000

4.3 Marketability

The newer model RT3300, which is basically the same robot as the RT3200 model but with a vision system, is priced at $60,000. Evaluating our budget/cost analysis, it shows that we will be able to provide the vision system we developed and offer it at a much lower price. Being that the development of the vision system with the Seiko D-TRAN RT3200 is already complete, it is possible to sell the vision system at a much lower cost than the $32,202 shown. Seiko has sold many RT3200 robots, which means that there are many potential RT3200 owners that would need vision system update. With our ability to offer our vision system at a much lower price than purchasing a RT3300 for that includes, companies can save a great deal of money. Evaluating the existing options of current RT3200 model owners, it appears that there is a possibility of a very profitable opportunity if our vision system is marketed properly. However, due to limited research and knowledge about the number of owners of the Seiko RT3200 robot, it is impossible to predict the size of the market for our product.

Chapter 5 ~ Meeting Expectations

5.1 Introduction

Our project has met all the specifications in the problem statement along with all of our primary objectives. The robot can now be controlled through Visual Basic, and with the addition of the vision system the robot can also scan for objects within the vision's workspace and perform specified task. The vision system has increased the capabilities and value to the Seiko robot arm. The Design Constraints are shown in section 6.2 and the Elements of Design are in section 6.3.

5.2 Design Constraints

Since our project is a continuation of last years design, we are limited to the approaches and direction in which we can take our design. Due to last year's project, we were limited to using Visual Basic as our programming language, which links the controller of the Seiko robot to the vision system. We were also constrained to the VisionBlox software to perform our image processing. The camera, frame grabber card, and the serial port were all hardware parts that we were limited to use for our project.

Another constraint to our project is the end-effector that was provided by the Electrical Engineering Department of UCR. The end-effector provided is very small and have no real life applicability. We will not be able get a new end-effector before the end of the 2002 Spring quarter, the demo for our project will be severely limited.

5.3 Elements of Design

The uses of robotics in the industrial field have long been an accepted and normal practice of the industry. There are many benefits of using robotics for production; such as safety, productive, reliable, and relatively cheap compared to human labor. For example, when working with dangerous chemicals or in unsafe environments, robot can be used to get the job done without endangering people's lives. Robots are also very productive for it can perform specified task at a very high speed without ever having to take a break. In most cases, machines will outperform human labor drastically in any mass production. With the advancement of robotics and technology, the robots used today in industry are extremely reliable and our experiences working with the Seiko RT3200 agrees with that. Although purchasing an industrial robot might seem quite expensive, once the robot is purchased the cost to maintain and reuse is relatively minimal. So in the long run, the use of robots for industrial applications will save companies tremendously in cost.

There will always be critics that will oppose the use of robots for the fear of losing all jobs to machines. It is true that machinery and robotics have taking over many of the production that used to be done by people, but by using robotics it has also made many things much more cheaper and available. So until something major event occurs, machinery and robots will continue to be integrated into our everyday lives and continue to grow.

Chapter 6 ~ Conclusions

6.1 Introduction

The problem statement of this project is to upgrade the Seiko RT3200 so that it can utilize vision by incorporating a Visual Basic interface using the software VisionBlox to implement with the Seiko robot arm. The robot will be able to recognize trained objects and perform a particular task with the object. We have succeeded in controlling the robot by using the Visual Basic user front and the vision system can recognize a trained image we specified. Using a trained image, we are able to get the Seiko robot to scan a particular workspace for the trained object. Then, once the trained object is found, we are able to allocate to the robot the location of the object and have it pick it up, and all of this can automated. Therefore, we have accomplished all the goals that we set out to accomplish and the rest will be left to the future seniors that will be working on the improvement of this project.

6.2 Design Expansion / Improvement Suggestions

There are many suggestions that we have for the improvement and expansion of this project in the future. The first is to mount the camera in a higher position so that the pictures captured by the camera will be in a better focus. As of now, the height of the object from the end-effector when the image is in focus is far too low, and when the end-effector tries to pick up a focused object, it cannot reach it. The alternative to this approach may be making a longer end-effector so that the end-effector can reach to a lower position, and therefore will not need to change the position of the camera. The second approach may be better because the calculation of the pixel to world coordinate system would stay relatively the same. Important note, if the camera and the distance between the platform is changed, then the area of the camera relative to the spatial area will also change, causing the pixel to world coordinate system calculation to change as well.

Another suggestion we have for the improvement of this project is to change the end-effector on the Seiko RT3200 robot arm to something much more useful. The end-effector we have now is able to pick up 2-4 mm wide objects only. We suggest buying a new end-effector that can actually be used to do something much more useful so a more impressive demonstration can use to showcase the usefulness of the project.

Other suggestions that future groups can also consider are: working in C instead of Visual Basic, incorporation the vision system for the Adaptec robot, storing the commands to read code through the serial port, and adding more features to the existing vision system. The reason for working in the computer language C is because the PX610 driver seems to be more compatible with C, and all its sample codes are in C, therefore it will be much easier to implement new features in the future. Also, vision system created for the Seiko robot should be easily incorporated into the Adaptec robot. Of course the commands to control the robot and camera location will be different, the general structure should be the same.

One of the additions that we were meaning to add but did not get to is getting rid

of the input commands of

DO COMM 1 78
DO OUNIT 1
DO IUNIT 1

When going from the control pendant to the Visual Basic program, these three commands must be first typed on the control pendant to get the controller to respond to commands sent through the serial port. This is a redundant procedure, and getting rid of it will please future users who want to send commands to the robot via the serial port. Finally, this project can be improved also just by adding more features and functionality to the existing vision system. The addition of edge detection, caliper, and blob tools of VisionBlox would be great addition to the existing vision system.

6.3 User's Manual

To start controlling the robot from the computer, first you have to type three commands from the teach pendant:

DO COMM 1 78
DO OUNIT 1
DO IUNIT 1

These inputs instructs the controller to read command lines through the serial port instead of the teach pendant. So all commands forth is assumed to be typed using the Visual Basic interface created.

Buttons:

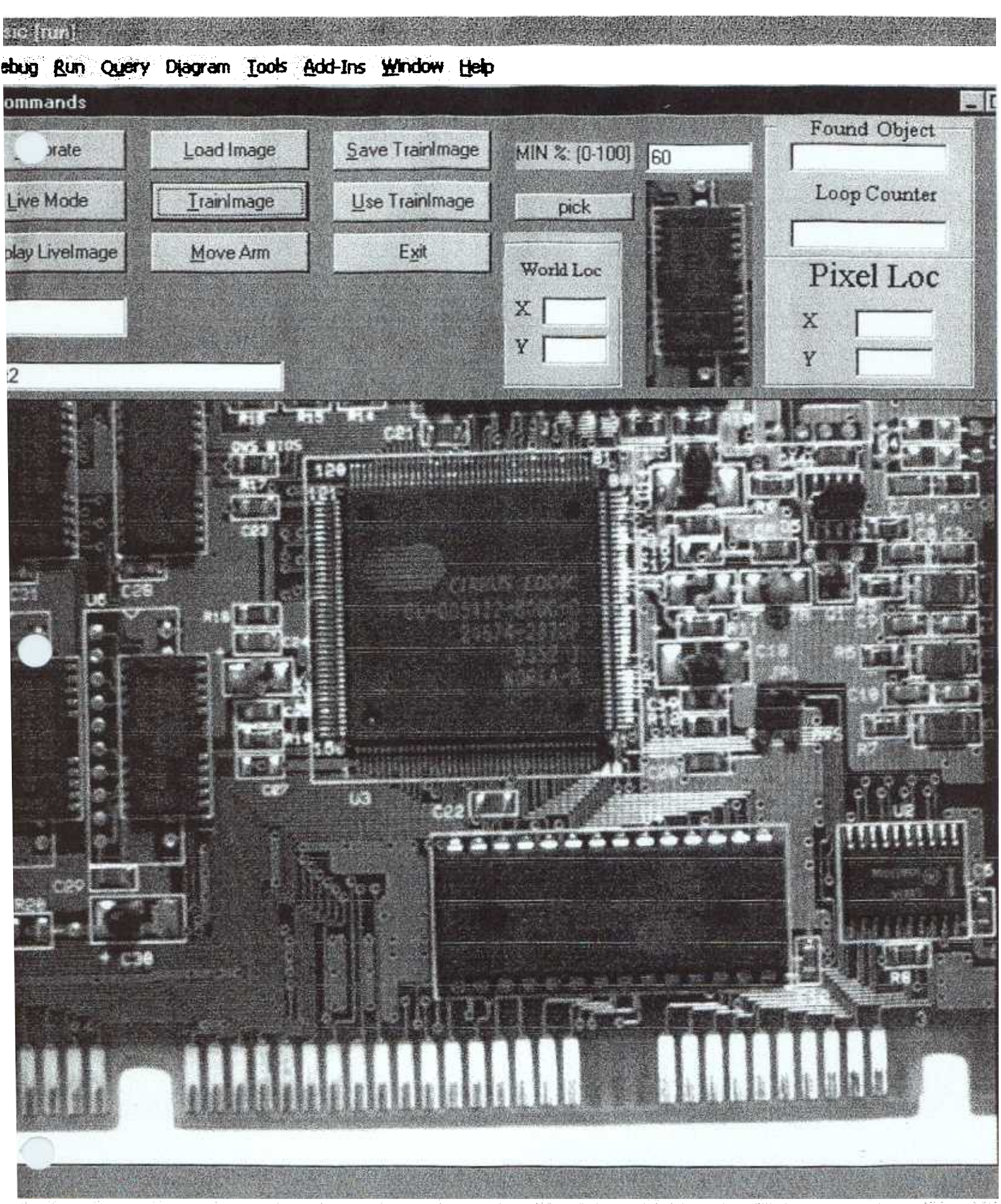
Calibrate – go to HOME position
Load Image – Loads a bitmap image onto the editable image screen
Save TrainImage – Saves the image currently on the edshpTrain of the FeatureFindTool
Pick – Calls the function that moves the robot to pick up an object
Live Mode – Performs the scanning and picking up the trained object
Display LiveImage – Goes to the live image screen and controller display
Move Arm – Moves the robot to a specified location
TrainImage – Trains a specific region of an image and the image is used for recognition
Use TrainImage – This checks if the train image is currently on the editable image display, and if so, the coordinates of the object is displayed
Exit – Quits the program

Textbox:

World Loc – Displays the location of the object with respect to the robot, these are coordinates that the robot uses to pick up the object
Pixel Loc – The location of the object with respect to the editable image

Loop Counter – Use to display the number of scans when in the live mode
Found Object – Returns true or false that the object is in the editable image
Minscore – Use to control how similar an object has to be with respect to the train image before it is considered the same object

Before performing any motion of the robot for the first time, the robot must be calibrated, and clicking on the "Calibrate" button on the user interface does this. Calibrating will place the robot to the HOME position.


ebug Run Query Diagram Tools Add-Ins Window Help
ommands
Load Image
Save TrainImage
MIN %: (0-100)
60
Found Object
Live Mode
TrainImage
Use TrainImage
pick
Loop Counter
Move Arm
Exit
World Loc
X
Y
Pixel Loc
X
Y
zhixu on 'coredump' (H:)
SeniorProject
Lastyear
Project1 - Microsoft Vis...

Appendix A: Parts List

Software:
Microsoft Visual Basic 6.0
VisionBlox
Darl

Hardware:
Seiko D-TRAN RT3200
Camera: Hitachi KP-M2
Personal Computer
Imagenation PX610 frame grabber
RS232 serial port
BNC video port

Appendix B: PX610 DLL Library Code

```
'Declarations for the PXDVNT.DLL library

Declare Sub pxPaintDisplay Lib "pxdvnt.dll" (ByVal hDC As Long, ByVal frm As Long, ByVal x As Lon
g, ByVal y As Long, ByVal dx As Long, ByVal dy As Long)
Declare Sub pxSetWindowSize Lib "pxdvnt.dll" (ByVal x As Long, ByVal y As Long, ByVal dx As Long,
 al dy As Long)
```

```
Global fgh As Long
Global frh As Long
Global counter As Long
Sub displayit()
    Dim x As Integer, y As Integer
    Dim dx As Integer, dy As Integer
    dx = (640& * Controller.HScroll1.Value) / 100&
    dy = (480& * Controller.HScroll1.Value) / 100&
    x = (640& - dx) * LiveImage.HScroll1.Value / 100&
    y = (480& - dy) * LiveImage.VScroll1.Value / 100&

    Call pxPaintDisplay(LiveImage.hDC, frh, x, y, dx, dy)
End Sub
```

```
''''''''''''''''''''''''''''''''''''''''''''''
 WPX5_NT.BAS
''''''''''''''''''''''''''''''''''''''''''''''
               Flags

Global Const GRAB_OFF = &H0
Global Const FIELDS = &H3
Global Const EITHER = &H3
Global Const GRAB_ON = &H3
Global Const FIELD0 = &H1
Global Const FIELD1 = &H2
Global Const IMMEDIATE = &H4
Global Const QUEUED = &H8
Global Const SINGLE_FLD = &H10
Global Const CACHE = &H20

               Trigger Types

Global Const RISING = &H3
Global Const FALLING = &H2
Global Const HIGH = &H1
Global Const LOW = &H0
Global Const DEBOUNCE = &H4

               Strobe Types
Global Const STROBE_0 = &H1
Global Const STROBE_1 = &H2
Global Const STROBE_OFF = &H4
Global Const STROBING = &H8
Global Const STROBE_GAP = &H10
Global Const STROBE_STOP = &H11
Global Const STROBE_TRIG = &H12
Global Const STROBE_NORMAL = &H13

Global Const SYNC_NORMAL = &H13
Global Const SYNC_OFF = &H4

               Error Numbers

Global Const ERR_NONE = &H0
Global Const ERR_NOT_VALID = &H1
Global Const ERR_CORRUPT = &H2
Global Const ERR_NO_PCI = &H4
Global Const ERR_BAD_IRQ = &H8
Global Const ERR_NO_ADDRESS = &H10
Global Const ERR_NO_DEVICES = &H20
Global Const ERR_GRAB_INVALID = &H80


               File I/O Return Values

Global Const SUCCESS = 0
Global Const FILE_EXISTS = 2
Global Const FILE_OPEN_ERROR =
Global Const BAD_WRITE = 4
Global Const BAD_READ = 5
Global Const WRONG_BITS = 8
Global Const BAD_FILE = 11
Global Const INVALID_FRAME = 12

               Sync Types
Global Const USER_SYNC = &H1
Global Const INTERNAL_SYNC = &H2
Global Const WEN_SYNC = &H4
Global Const SINGLE_FIELD_SYNC = &H10
Global Const AUTOMATIC_SYNC = &H80

               Configuration Values
Global Const PXC_CACHE = &H1
Global Const PXC_NONINTERLACE = &H2
```

```
Global Const PXC_BUS = &HC0
Global Const PXC_PCI = &H0
Global Const PXC_104_PLUS = &H40
Global Const PXC_COMPACT_PCI = &H80
Global Const PXC_VIDEO_DRIVE = &H100
Global Const PXC_H_CROP = &H100
Global Const PXC_V_SCALE = &H100
Global Const PXC_V_CROP = &H800
Global Const PXC_H_SCALE = &H800
Global Const PXC_STROBES = &H200
Global Const PXC_WEN_SYNC = &H400
Global Const PXC_CUSTOM_HW = &HF000


                Setup Functions

Declare Function InitLibrary Lib "wpx5_nt.dll" () As Long
Declare Sub ExitLibrary Lib "wpx5_nt.dll" ()


                Frame Data Functions

Declare Function AllocateBuffer Lib "wpx5_nt.dll" (ByVal dx As Long, ByVal dy As Long, ByVal bits
 As Long) As Long
Declare Function AllocateAddress Lib "wpx5_nt.dll" (ByVal address As Long, ByVal dx As Long, ByVa
l dy As Long, ByVal bits As Long) As Long
Declare Sub FreeFrame Lib "wpx5_nt.dll" (ByVal frh As Long)
Declare Function FrameWidth Lib "wpx5_nt.dll" (ByVal frh As Long) As Long
Declare Function FrameHeight Lib "wpx5_nt.dll" (ByVal frh As Long) As Long
Declare Function FrameBits Lib "wpx5_nt.dll" (ByVal frh As Long) As Long
Declare Function FrameBuffer Lib "wpx5_nt.dll" (ByVal frh As Long) As Long
'FrameAddress NOT supported on NT


                Frame Grabber Allocation Functions

Declare Function AllocateFG Lib "wpx5_nt.dll" (ByVal n As Long) As Long
Declare Sub FreeFG Lib "wpx5_nt.dll" (ByVal fgh As Long)


                Grab Functions

Declare Function GrabContinuous Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal frh As Long, ByVal ac
tive As Long) As Long
Declare Function Grab Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal frh As Long, ByVal flags As Lon
g) As Long
Declare Function GrabTriggered Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal frh As Long, ByVal del
time As Long, ByVal flags As Long) As Long
Declare Function ReadCache Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal frh As Long, ByVal flags A
s Long) As Long
Declare Function GrabToCache Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal flags As Long) As Long
Declare Function CacheTriggered Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal deltime As Long, ByVa
l flags As Long) As Long
Declare Function Wait Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal flags As Long) As Long
Declare Function WaitVB Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long
Declare Function IsFinished Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal handle As Long) As Long
Declare Sub KillQueue Lib "wpx5_nt.dll" (ByVal fgh As Long)


                Frame Grabber State Functions

Declare Function SetFineGain Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal gain As Long, ByVal flag
s Long) As Long
Declare Function SetGainRange Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal range As Long, ByVal fl
ags As Long) As Long
Declare Function SetOffset Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal offset As Long, ByVal flag
s As Long) As Long
Declare Function SetTriggerType Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal mode As Long) As Long
Declare Function SetLUT Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal first_address As Long, ByVal
length As Long, lutarray As Long) As Long
```

```
Declare Function SetCamera Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal camera As Long, ByVal flag
s As Long) As Long
Declare Sub ResetFG Lib "wpx5_nt.dll" (ByVal fgh As Long)
Declare Function SetImageSize Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal resx As Long, ByVal res
y As Long, ByVal x0 As Long, ByVal y0 As Long, ByVal dx As Long, ByVal dy As Long, ByVal bits As
Long) As Long
Declare Function SetCompare Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal n As Long) As Long
Declare Function SetFieldCount Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal x As Long) As Long
'SetCurrentWindow NOT supported on NT
Declare Function SetStrobePolarity Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal s As Long, ByVal p
As Long) As Long
Declare Function SetFieldSize Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal resx As Long, ByVal res
y As Long, ByVal x0 As Long, ByVal y0 As Long, ByVal dx As Long, ByVal dy As Long, ByVal bits As
Long) As Long
Declare Function SetVideoFormat Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal field_length As Long,
ByVal blank_length As Long, ByVal flags As Long) As Long


                Frame Grabber Info Functions

Declare Function ReadRevision Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long
Declare Function ReadProtection Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long
Declare Function HaveCache Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long
Declare Function VideoType Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long
Declare Function CheckGreater Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long
Declare Function CheckEqual Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long
Declare Function GetTriggerType Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long
Declare Function GetFineGain Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long
Declare Function GetGainRange Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long
Declare Function GetOffset Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long
Declare Function GetCamera Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long
Declare Function GetLUT Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal first_address As Long  ByVal
length As Long, lutarray As Long) As Long
Declare Function CheckError Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long
Declare Function GetFieldCount Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long
Declare Function GetFieldLength Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long
Declare Function ReadConfiguration Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long


                Data Manipulation Functions

Declare Sub GetRectangle Lib "wpx5_nt.dll" (ByVal frh As Long, buf As Byte, ByVal x0 As Long, ByV
al y0 As Long, ByVal dx As Long, ByVal dy As Long)
Declare Sub PutRectangle Lib "wpx5_nt.dll" (buf As Byte, ByVal frh As Long, ByVal x0 As Long, ByV
al y0 As Long, ByVal dx As Long, ByVal dy As Long)
Declare Sub GetRow Lib "wpx5_nt.dll" (ByVal frh As Long, buf As Byte, ByVal row As Long)
Declare Sub PutRow Lib "wpx5_nt.dll" (buf As Byte, ByVal frh As Long, ByVal row As Long)
Declare Sub GetColumn Lib "wpx5_nt.dll" (ByVal frh As Long, buf As Byte, ByVal row As Long)
Declare Sub PutColumn Lib "wpx5_nt.dll" (buf As Byte, ByVal frh As Long, ByVal row As Long)
Declare Function ReadBin Lib "wpx5_nt.dll" Alias "ReadBinA" (ByVal frh As Long, ByVal fname As St
ring) As Long
Declare Function WriteBin Lib "wpx5_nt.dll" Alias "WriteBinA" (ByVal frh As Long, ByVal fname As
String, ByVal overwrite As Long) As Long
Declare Function ReadBMP Lib "wpx5_nt.dll" Alias "ReadBMPA" (ByVal frh As Long, ByVal fname As St
ring) As Long
Declare Function WriteBMP Lib "wpx5_nt.dll" Alias "WriteBMPA" (ByVal frh As Long, ByVal fname As
String, ByVal overwrite As Long) As Long

                Digital I/O functions -- strobes and syncs
Declare Function FireStrobe Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal command As Long) As Long
Declare Function GetStrobeState Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long
Declare Function GetSyncType Lib "wpx5_nt.dll" (ByVal fgh As Long) As Long
Declare Function SetDrivePolarity Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal vdrive As Long, ByV
al vdrive As Long) As Long
Declare Function SetDriveType Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal mode As Long) As Long
Declare Function SetStrobePeriods Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal t1 As Long, ByVal t
2 As Long, ByVal t3 As Long) As Long
Declare Function SetStrobeType Lib "wpx5_nt.dll" (ByVal fgh As Long, ByVal mode As Long) As Long
```

Appendix C: Visual Basic Form Codes

```
iveImage - 1


 ivate Sub Form_Load(
    Dim x As Integer
    Dim y As Integer

    '**Fixed Window Size**
    'This requires moving the scrollbars

    LiveImage.ScaleMode = 3
    VScroll1.Top = 0
    VScroll1.Left = 640
    VScroll1.Height = 512
    HScroll1.Top = 512
    HScroll1.Left = 0
    HScroll1.Width = 640
    Call pxSetWindowSize(0, 0, 640, 512)
    'We calculate the new window size in twips because
    'screen coordinates are in twips.
    LiveImage.ScaleMode = 1
    x = VScroll1.Left + VScroll1.Width
    y = HScroll1.Top + HScroll1.Height
    'account for the window border size
    x = x + (LiveImage.Width - LiveImage.ScaleWidth)
    y = y + (LiveImage.Height - LiveImage.ScaleHeight)
    LiveImage.Width = x
    LiveImage.Height = y

    LiveImage.Timer1.Interval = 10
nd Sub

rivate Sub Form_Paint()
    Call displayit
nd Sub

    ate Sub HScroll1_Change()
    Call displayit

    Sub

rivate Sub Timer1_Timer()
    counter = counter + 1
    tmp = Grab(fgh, frh, 0)
    If (tmp = 0) Then
        Debug.Print "Grab failed"
    End If
    Call displayit
nd Sub

rivate Sub VScroll1_Change(
    Call displayit
nd Sub
```

```
ontroller - 1

rivate Sub Command1_Click()
    Call Commands.GetImage
nd Sub



rivate Sub Form_Load()
    If (InitLibrary() = 0) Then
        Debug.Print "init fail"
        End
    End If

    fgh = AllocateFG(-1)
    If (fgh = 0) Then
        ExitLibrary
        Debug.Print "frame grabber fail"
        End
    End If

    frh = AllocateBuffer(640, 480, 8)
    If (frh = 0) Then
        FreeFG (fgh)
        ExitLibrary
        Debug.Print "buffer fail"
        End
    End If

    tmp = SetImageSize(fgh, 640, 256, 0, 4  640  480  8)
    If (tmp = 0) Then
        Debug.Print "SetImageSize failed"
    End If
    LiveImage.Show
 d Sub

    ate Sub Form_Unload(Cancel As Integer)
    LiveImage.Timer1.Interval = 0
    Unload LiveImage

    FreeFrame (frh)
    FreeFG (fgh)
    ExitLibrary

 id Sub

rivate Sub grange_Click(Index As Integer)
    tmp = SetGainRange(fgh, Index, 0)
nd Sub

rivate Sub gslide_Change()
    tmp = SetFineGain(fgh, gslide.Value, 0)
nd Sub

rivate Sub HScroll1_Change()
    Call displayit
nd Sub

rivate Sub Label4_Click

nd Sub

rivate Sub offslide_Change()
    tmp = SetOffset(fgh, offslide.Value  0)

   Sub

 ivate Sub Open_Click()
    stop_click
    CMDialog1.FileName = ""
    CMDialog1.Action = 1
    If (CMDialog1.FileName <> ""  Then
```

```
Controller - 2

        Screen.MousePointer = 11
        If ReadBMP(frh, CMDialog1.FileName) Then
            Screen.MousePointer = 0
            MsgBox "Could Not Read File", 48, "Error"
        End If
        Screen.MousePointer = 0
    End If
    LiveImage.Refresh
End Sub

Public Sub play_Click()
    LiveImage.Timer1.Interval = 10
    counter = 0
    t1 = Timer


Private Sub save_Click()
    stop_click
    CMDialog1.FileName = ""
    CMDialog1.Action = 2
    If (CMDialog1.FileName <> "") Then
        Screen.MousePointer = 11
        If WriteBMP(frh, CMDialog1.FileName, 1) Then
            Screen.MousePointer = 0
            MsgBox "Could Not Write File", 48, "Error
        End If
        Screen.MousePointer = 0
    End If
    'Call WriteBMP(frh, "c:\ee175AB\test.bmp", 1)
End Sub

Private Sub stop_click()
    LiveImage.Timer1.Interval = 0
End Sub


Private Sub cmdQuit_Click()
    LiveImage.Timer1.Interval = 0
    Unload LiveImage
    Unload Controller
    Load Commands
    Commands.Visible = True
    Commands.WindowState = 0

    FreeFrame (frh)
    FreeFG (fgh)
    ExitLibrary
End Sub
```

```
Option Explicit

' Defining global variable of the form

Private mintResultNumber As Integer
Private counter_x As Integer
Dim loop_counter As Integer
Private flag As Integer


Private Sub Command1_Click()
    Call GetImage
End Sub


Private Sub cmdLoad_Click()

      Loads a saved bitmap image and sets to the editable image screen
      When clicked, a file must be selected or an error occurs. A if statement can be used
      but we consider this not the most important issue right now so disregard for now

    Commands.CommonDialog1.DialogTitle = "File Open"
    Commands.CommonDialog1.Filter = "*.bmp"
    Commands.CommonDialog1.FileName = "*.bmp"
    Commands.CommonDialog1.ShowOpen

     load the image information from the FeatureFindSample.INI file

    Commands.edshpTrain.Load "h:\SeniorProject\LastYear\FeatureFindSample.INI", "TrainingShape"
    Commands.edshpInspection.Load "h:\SeniorProject\LastYear\FeatureFindSample.INI", "InspectionS
hape"
    Commands.FeatureFindTool.Load "h:\SeniorProject\LastYear\FeatureFindSample.INI", "Featurefind
    "
    Commands.FeatureFindTool.Refresh

End Sub


Private Sub cmdMOVE_Click()

    ' Moves robot to one location right now for testing purposes. Need to be incorporated later
 allow
    ' user a list of coordinate choices or allow user to input location

    MSComm1.Output = "DO SPEED 50" + vbCrLf
    Call Move1(380, 170, -120, 0)

End Sub


Private Sub cmdPICK_Click()

   Call pick(True)

End Sub


Private Sub cmdSave_Click()

    'Saves the current trained image to a file  Filename "image1.Dat" is used at the default name


    Commands.CommonDialog2.DialogTitle = "File Save"
    Commands.CommonDialog2.Filter = "*.Dat"
    Commands.CommonDialog2.FileName = "image1.Dat"
    Commands.CommonDialog2.Action = 2
    If (Commands.CommonDialog2.FileName <> "") Then
        Screen.MousePointer = 11
```

```
ommands

        If WriteBMP(frh, Commands.CommonDialog2.FileName, 1) Then
            Screen.MousePointer = 0
            MsgBox "Could Not Write File", 48, "Error"
        End If
        Screen.MousePointer = 0
    End If

    'saves image and feature find information

    Commands.FeatureFindTool.FeatureFileName = Commands.CommonDialog2.FileName
    Commands.edshpTrain.save "h:\SeniorProject\LastYear\FeatureFindSample.INI", "TrainingShape"
    Commands.edshpInspection.save "h:\SeniorProject\LastYear\FeatureFindSample.INI", "InspectionS
ape"
    Commands.FeatureFindTool.save "h:\SeniorProject\LastYear\FeatureFindSample.INI", "Featurefind
ool"



ivate Sub commandUse_Click()

    ' Compares the features of the trained image with the image being displayed on the editable i


    Commands.FeatureFindTool.Use
    Commands.FeatureFindTool.MinimumScore = Commands.txtMinScore
    Commands.CalXValue.Text = Commands.FeatureFindTool.Found



rivate Sub commandTrain_Click()

    ' Trains the image in the train shape box  Rectangular blue shaped on the editable image. Cli
nd
      hold to drag shape

    Commands.FeatureFindTool.Train
    Commands.commandUse.Enabled = True
    Commands.edshpInspection.Visible = True
    Commands.FeatureFindTool.Refresh

d Sub


rivate Sub disp_Click(Index As Integer)

      Displays the live image on the Liveimage window.

    Commands.WindowState = 1
    Controller.Show
    LiveImage.Show

nd Sub


rivate Sub EdibleImage_Paint()


    'Controls what to display on the edible image

    'Commands.EdibleImage.Picture = "h:\SeniorProject\LastYear\abc.bmp"

    If (Commands.CommonDialog1.FileName = "") Then
        Commands.EdibleImage.Picture = "h:\SeniorProject\LastYear\abc.bmp"

    ElseIf (flag = 1) Then
        Commands.EdibleImage.Picture = "h:\SeniorProject\LastYear\abc.bmp"
        Commands.EdibleImage.Refresh
        flag = 0
```

```
    Else
        Commands.EdibleImage.Picture = "h:\SeniorProject\LastYear\abc.bmp"
        Commands.EdibleImage.Picture = Commands.CommonDialog1.FileName
    End If

    Sub


Private Sub FeatureFindTool_Use

      When features desired are found on the edible image  this controls what is displayed

    If Commands.FeatureFindTool.NumberOfResults   0 Then
        mintResultNumber = 1
        DisplayResultInfo
    Else
        mintResultNumber = 0
        DisplayResultInfo
    End If

    Sub

Private Sub Form_Load

      Loads the form, initializes and sets parameter before anything are done

  Dim CommPort As String, Handshaking As String, Settings As String
  Dim counter_x As Integer

    ' intializes these global variables
    counter_x = 0
    counter = 0
    flag = 0

    ' need to be specified for featurefindtool to work
    Commands.FeatureFindTool.hInspectionImage = EdibleImage
    Commands.FeatureFindTool.hInspectionShape = edshpInspection
    Commands.FeatureFindTool.hTrainingImage = Commands.FeatureFindTool.hInspectionImage
    Commands.FeatureFindTool.hTrainingShape = edshpTrain

    ' image shown when program start at beginning
    Commands.EdibleImage.Picture = "h:\SeniorProject\LastYear\abc.bmp"
    Commands.edshpTrain.Load "h:\SeniorProject\LastYear\abc.INI", "TrainingShape"
    Commands.edshpInspection.Load "h:\SeniorProject\LastYear\abc.INI", "InspectionShape"
    Commands.FeatureFindTool.Load "h:\SeniorProject\LastYear\abc.INI", "FeaturefindTool"
    Commands.FeatureFindTool.Refresh

    ' to show when program is running
    Commands.edshpTrain.Visible = True
    Commands.edshpInspection.Visible = True
    Commands.commandUse.Enabled = True
    Commands.commandTrain.Enabled = True

    'initial value used, can be changed when program is running
    Commands.txtMinScore = "75"

      serial port settings, very important

    CommPort = 1
    Settings = "9600,N,8,1"

    Use COM1.
    MSComm1.CommPort = CommPort

     9600 baud, no parity, 8 data, and 1 stop bit
    MSComm1.Settings = Settings

     Handshaking
    'MSComm1.Handshaking = comRTS
```

```
    ' Open Port
    If MSComm1.PortOpen = False Then
        MSComm1.PortOpen = True
    End If



Private Sub DisplayResultInfo()

    'Displays the results of when using the "Use" button to the screen

    If mintResultNumber = 0 Then

        Commands.XValue.Text = ""
        Commands.YValue.Text = ""

        Commands.ResultLabel.Enabled = False
        Commands.FrameCoordinates.Enabled = False
    Else
        Commands.ResultLabel.Enabled = True
        Commands.FrameCoordinates.Enabled = True

        Commands.XValue.Text = Commands.FeatureFindTool.ResultX(mintResultNumber)
        Commands.YValue.Text = Commands.FeatureFindTool.ResultY(mintResultNumber)


        'Point at the origin of the area where the feature was found when the Use method was last
invoked

        Commands.Text1 = Commands.FeatureFindTool.ResultPoint(mintResultNumber)
        If

    Sub


Private Sub cal_Click(Index As Integer)

     Output calibration command HOME  Calibrates the robots

    MSComm1.Output = "HOME"
    MSComm1.Output = vbCrLf
    Call delay(2)



Private Sub Exit_Click(Index As Integer)

      Close Port

    If MSComm1.PortOpen <> False Then
        MSComm1.PortOpen = False
    End If
    ' Exit Program
    End



Sub Move1(x As Double, y As Double, z As Double  R As Double)

    Dim buffer As String
    Dim destination As String

    'manually move arm for given destination
    Dim T(1 To 4) As String
    T(1) = CStr(x)
    T(2) = CStr(y)
```

```
T(3) = CStr(z)
T(4) = CStr(R)

'Define point
destination = T(1) + " " + T(2) + " " + T(3) + " " + T(4)
buffer = "DO T1 = " + destination
MSComm1.Output = buffer + vbCrLf
Call delay(7)
MSComm1.Output = "DO MOVE T1" + vbCrLf


Sub


Sub delay(c As Integer)

'Delay for specified times  Used quite often to make sure robot and vision can be worked toge


For counter = 1 To c * 1000000
Next counter

End Sub


Sub pick(y As Boolean)

  To pick object if found, or increment to next area if not found  Loop is under the RunProg(

  function to scan workspace of robot.

Dim screen_x As Double
Dim screen_y As Double
Dim darl_x As Integer
Dim darl_y As Integer
Dim A As String
Dim move_x As Integer
Dim B As String
Dim move_x2 As Integer
Dim ndarl_x As Integer

'clear robot memory bank
MSComm1.Output = "DO CLEAR" & vbCrLf
    Call delay(5)

y = Commands.FeatureFindTool.Found

If y = True Then

    'get pixel location
    screen_x = Commands.FeatureFindTool.ResultX(mintResultNumber)
    screen_y = Commands.FeatureFindTool.ResultY(mintResultNumber)

    darl_y = CInt((64.516 - (screen_x / 6.124) ) 'convert pixel location to real cooridate
    darl_x = CInt((screen_y / -6.124) + 387)   'convert pixel location to real cooridate
    ndarl_x = darl_x + counter_x

    Commands.txtWorldX.Text = ndarl_x
    Commands.txtWorldY.Text = darl_y

    'set darl command into string
    A = "DO T1 = " & ndarl_x & ". " & darl_y & "  -100  0

    Commands.cmdA.Text = A

    'set robot movement speed
    MSComm1.Output = "DO SPEED 300" & vbCrLf
    Call delay(1)

    Commands.cmdA.Text = A
```

```
      Call delay(1)
      MSComm1.Output = A
      MSComm1.Output = vbCrLf
      Call delay(5)
      'open end-effector
      MSComm1.Output = "DO OUTPUT + OG6" & vbCrLf
      Call delay(40)
      'move to found object
      MSComm1.Output = "DO MOVE T1" & vbCrLf
      Call delay(40)
      'end-effector go down
      MSComm1.Output = "DO OUTPUT + OG7" & vbCrLf
      Call delay(40)
      'close end-effector
      MSComm1.Output = "DO OUTPUT - OG6" & vbCrLf
      Call delay(40)
      'end-effector go up
      MSComm1.Output = "DO OUTPUT - OG7" & vbCrLf
      Call delay(40)
      'set the destination that want to drop the object
      MSComm1.Output = "DO T3 = -17. 473. -120. 0." & vbCrLf
      Call delay(5)
      'go to the destination that want to drop the object
      MSComm1.Output = "DO MOVE T3" & vbCrLf
      Call delay(40)
      'drop object
      MSComm1.Output = "DO OUTPUT + OG6" & vbCrLf
      Call delay(40)

      'example for last demo
      'MSComm1.Output = "DO T4 = 369. 34. -91. 0." & vbCrLf
      'Call delay(5)
      'MSComm1.Output = "DO MOVE T4" & vbCrLf
      'Call delay(40)
      'MSComm1.Output = "DO T5 = 369. 34. -96. 0." & vbCrLf
      'Call delay(5)
      'MSComm1.Output = "DO MOVE T5" & vbCrLf
      'Call delay(40)
      'MSComm1.Output = "DO OUTPUT + OG6" & vbCrLf
      'Call delay(40)
      counter_x = 0
   Else
      'keep increment until find a object
      counter_x = counter_x + 10
      MSComm1.Output = "DO OUTPUT - OG7" & vbCrLf
      Call delay(40)
      Commands.Text1 = counter_x
      move_x = 300
      move_x2 = move_x + counter_x
      B = "DO T10= " & move_x2 & ". 0. 0. 0."
      MSComm1.Output = B
      MSComm1.Output = vbCrLf
      Call delay(5)
      MSComm1.Output = "DO MOVE T10" & vbCrLf
      Call delay(5)

         End If
   Call delay(15)
End Sub



   lic Sub GetImage()

     Not used

   Dim tmp As Integer

   tmp = Grab(fgh, frh, 0)
   Call WriteBMP(frh, "c:\ee175AB\temp.bmp", 1
```

```
Commands

End Sub


Private Sub RunProg_Click(Index As Integer)

    Automation of the vision system, utilizes the pick function

  Dim temp As Integer

  ' loop until object is found
  Do

      ' setting live image to abc.bmp
      Call Controller.play_Click
      LiveImage.Timer1.Interval = 0
      temp = Grab(fgh, frh, 0)
      Call WriteBMP(frh, "h:\SeniorProject\LastYear\abc.bmp", 1)
      Commands.EdibleImage.Picture = "h:\SeniorProject\LastYear\abc.bmp"

      ' saves abc.bmp info to abc.ini file
      Commands.FeatureFindTool.FeatureFileName = "h:\SeniorProject\LastYear\abc.Dat"
      Commands.edshpTrain.save "h:\SeniorProject\LastYear\abc.INI", "TrainingShape"
      Commands.edshpInspection.save "h:\SeniorProject\LastYear\abc.INI", "InspectionShape"
      Commands.FeatureFindTool.save "h:\SeniorProject\LastYear\abc.INI", "FeaturefindTool"

      'load to be displayed and use
      Commands.edshpTrain.Load "h:\SeniorProject\LastYear\abc.INI", "TrainingShape"
      Commands.edshpInspection.Load "h:\SeniorProject\LastYear\abc.INI", "InspectionShape"
      Commands.FeatureFindTool.Load "h:\SeniorProject\LastYear\abc.INI", "FeaturefindTool"

      Commands.FeatureFindTool.Refresh
      Commands.EdibleImage.Refresh
      Controller.Refresh
      LiveImage.Refresh

      ' check if trained image is found by calling the use function
      Call commandUse_Click
      Call pick(Commands.FeatureFindTool.Found)
      Call delay(20)

      ' setting the edible image to abc.bmp, loop counter
      flag = 1
      Commands.WindowState = 1
      loop_counter = loop_counter + 1
      Commands.CalYValue.Text = loop_counter


  Loop While (Commands.FeatureFindTool.Found = False And loop_counter < 10)

  ' reseting variable after loop
  counter_x = 0
  loop_counter = 0
  Commands.WindowState = 0
```

Appendix D: Technical Specifications

KP-M2U Specification (EIA)

- **Sensor pick up device:** 1/2" (KP-M2) Interline transfer CCD
  1/3" (KP-M3) interline transfer CCD

  Sensing scanning area: 6.45 (H) x 4.84 (V) mm (KP-M2)
  4.88 (H) x 3.09 (V) mm (KP-M3)

  Total number of pixels: 811 (H) x 508 (V)

  Number of effective pixels: 768 (H) x 494 (V)

  Pixel pitch: 8.4 (H) x 9.8 (V) µm (KP-M2)
  6.35 (H) x 7.4 (V) µm (KP-M3)

  Spectral sensitivity:

  Scanning system: 2:1 interlaced / 525 lines / 60 fields per second / 30 frames per second or non-interlaced (only in external sync mode or with modification)

  Scanning frequency: 15.734 kHz (H), 59.94Hz (V)

  Signal standard: EIA Monochrome

  Sync system: Internal or external (automatic switching)

  External sync input: HV/VD 2 to 6 volts p-p input impedance 1kohm, frequency deviation ±1%, negative polarity

  Resolution: 570 (H), 485 (V) TV lines

  Standard sensitivity: 400 lux @ f4 (3200K)

  Minimum sensitivity: 0.5 lux at f1.4 (with AGC on, gamma 0.45, without IR cut filter)

  Electronic Shutter: off, 1/100th, 1/125th, 1/250th, 1/500th, 1/1,000th, 1/2,000th, 1/4,000th, 1/10,000th (selectable by external switch)

  Gamma correction: 0.45 or 1.0 (selectable by internal switch)

  Signal to noise ratio: 56dB

  AGC: Normal or AGC (selectable by internal switch)

Video output: VS 1.0 volt p-p 75 ohms unbalanced. (0.7 volts p-p video and 0.3 volts p-p sync negative polarity)

Integration mode: Field or frame integration (selectable by external switch)

Field on demand function: Switchable on/off by internal switch

Restart reset operation: On/Off selectable by component change

Lens mount: C

- **Flange focal distance:** 7.526 mm
- **Ambient temperature and humidity:**
  Operating: -10° C to +60°C, RH 90% or less
  Storage: -20° C to +60°C, RH 70% or less

Resistance to vibration: 9G max (10 to 60Hz amplitude constant at 0.98mm) 7G constant (60 to 150Hz amplitude variable), 10 to 150Hz (sweeping for one minute, each of X,Y and Z directions for 30 minutes)

Dimensions: 44 (W) x 29 (H) x 72 (D) mm

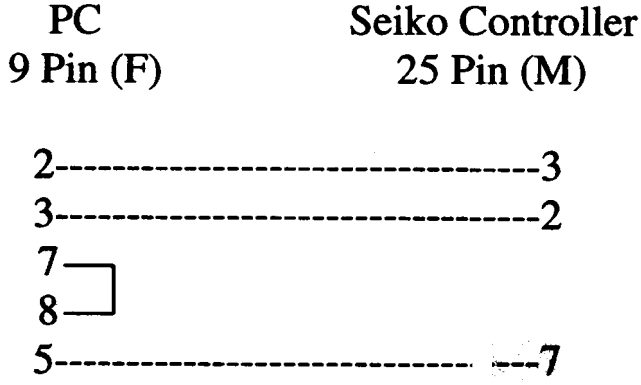
Weight: 120 grams (approx)

Supply voltage: 12 volts dc ±1 volt

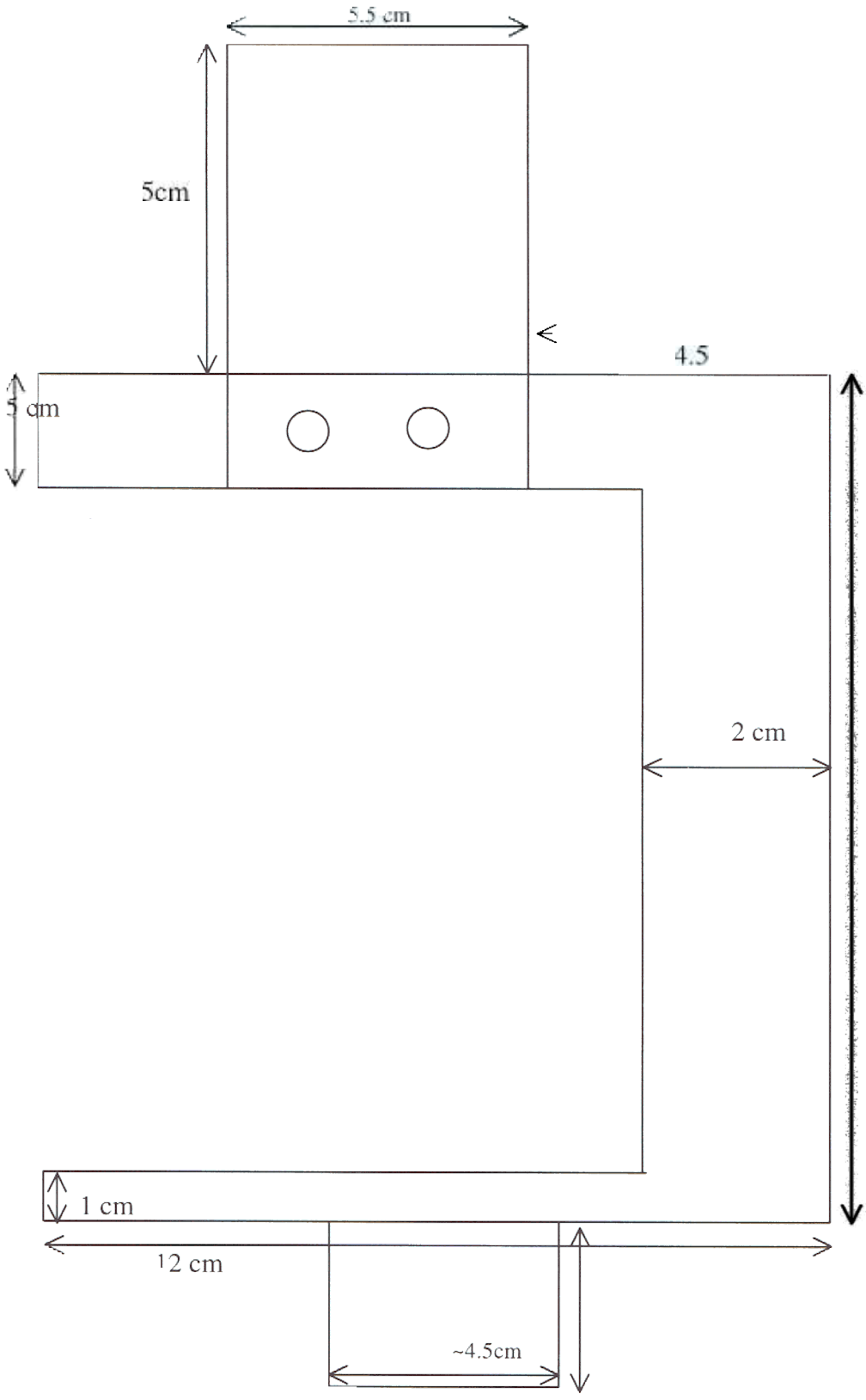
Current consumption: 180 mA (approx)

Composition: Camera head (with IR cut filter), C mount cap, operation manual

Seiko Robot RS-232 serial cable pin-outs



Camera mount design



References

[1] DARLfour Programming Guide. Seiko Instruments, Torrance, CA 1987

[2] DARL Talk. Version 2.02A, Seiko Instruments, Torrance, CA 1995.

[3] "Imaging Components: VisionBlox Imaging Software," Vision 1, http://www.vision1.com/software/vbloxdesc.shtml (current June 7, 2001).

[4] "KPM-2 Camera," Premier Electronics, http://www.premierelect.co.uk/kpm2.html (current June 7, 2001).

[5] Seiko RT3200 Installation and Operation Manual. ed., Seiko Instruments, Torrance, CA 1987.

[6] VisionBlox Training Manual. Version 3.0, Integral Vision Inc., Farmington Hills, MI 1998.

[7] VisionBlox Reference Manual. Version 3.0, Integral Vision Inc., Farmington Hills, MI 1998.

[8] G. Perry, Teach Yourself Visual Basic 5 in 24 Hours. 1st ed., Indianapolis, Indiana Sams Publishing, 1997.

[9] J.J. Craig, Introduction to Robotics. 2nd ed., Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1989.

[10] H.M. Deitel, P.J. Deitel, and T.R. Nieto, Visual Basic 6 How to Program. New Jersey: Prentice Hall, 1999.

[11] C. Bertell, H. Nguyen, and K. Wassink, Final Report of Senior Design Project Winter/Spring 2001, 2001.

Acknowledgement

We would like to thank Dan Giles for the all the help and guidance he has assisted us with all throughout. We also like to thank Dr. Gerardo Beni for keeping us on track with our report by making us send him weekly reports. Jojo, thanks for helping us prepare for the presentation, it really helped us from getting flustered when we presented in front of everyone. Steve, thanks for lending us the Visual Basic book. Also, thanks to Joseph for providing the us with the video camera so we can tape our demonstration and being our cameraman too.